Modèle de Sélection de Caractéristiques pour les Données Massives

Chelly Dagdia, Zaineb; Zarges, Christine; Beck, Gael; Lebbah, Mustapha

Published in:
15ème édition de l'atelier Fouille de Données Complexes

Publication date:
2018

Citation for published version (APA):
Chelly Dagdia, Z., Zarges, C., Beck, G., & Lebbah, M. (2018). Modèle de Sélection de Caractéristiques pour les Données Massives: Méthode de sélection de caractéristiques pour les données massives. In *15ème édition de l'atelier Fouille de Données Complexes: FDC*

Modèle de Sélection de Caractéristiques pour les Données Massives

Zaineb Chelly Dagdia[*,**] Christine Zarges[*]
Gael Beck[***], Mustapha Lebbah[***]

[*]Department of Computer Science, Aberystywth University, United Kingdom
[**]LARODEC, Institut Supérieur de Gestion de Tunis, Tunisia
{zaineb.chelly, c.zarges}@aber.ac.uk,
[***]Computer Science Laboratory (LIPN), University Paris-North - 13, Villetaneuse, France
{beck, mustapha.lebbah}@lipn.univ-paris13.fr

Résumé. La théorie des ensembles approximatifs est une approche pertinente pour la sélection des caractéristiques. Toutefois, cette dernière a un coût computationnel important et une application plus adaptée aux jeux de données de taille limitée. Par conséquent, dans cet article, nous présentons un algorithme distribué et scalable basé sur la théorie des ensembles approximatifs pour le prétraitement des données massives. Nos résultats expérimentaux montrent l'efficacité de notre solution sans perte d'information significative.

1 Introduction

Les données massives (ou Big Data en Anglais) surviennent souvent avec de nombreux défis liés aux prétraitement des données et spécifiquement à la sélection des caractéristiques Labrinidis et Jagadish (2012). Formellement, la tâche de la sélection des caractéristiques vise à déterminer un sous-ensemble minimal de caractéristiques à partir d'une représentation d'origine tout en conservant plus ou moins la même qualité. Cette tâche est éventuellement difficile suite au grand espace de recherche reflétant le grand nombre combinatoire de toutes les combinaisons possibles des caractéristiques Fan et Bifet (2013). Ainsi, afin de palier à cette problématique, un certain degré de réduction de caractéristiques s'avère nécessaire, par conséquent, une méthode efficace de sélection de caractéristiques est requise.

La Théorie des Ensembles Approximatifs (TEA) Pawlak et Skowron (2007); Pawlak (2012) est une approche performante pour la sélection des caractéristiques Thangavel et Pethalakshmi (2009). Cependant, la plupart des algorithmes basés sur cette théorie sont des algorithmes séquentiels, coûteux et ne peuvent traiter que de petits jeux de données. Ceci est expliqué par la nécessité de générer toutes les combinaisons possibles des caractéristiques, les traiter pour finalement sélectionner l'ensemble minimal des caractéristiques les plus pertinentes. Cependant, vu que le nombre des caractéristiques accroit d'une manière exponentielle dans le contexte des données massives cette tâche devient difficile. Cela nous amène à présenter dans cet article un nouvel algorithme basé sur la théorie des ensembles approximatifs pour un prétraitement de données à grande échelle. Notre nouvel algorithme est caractérisé par une implémentation

distribuée basée sur l'écosystème Scala/Apache Spark Shanahan et Dai (2015). Cette méthode est une première contribution qui s'inscrit dans le cadre d'un projet du programme H2020 de la bourse Marie Sklodowska-Curie, accord Numéro 702527.

2 Théorie des Ensembles Approximatifs

La Théorie des Ensembles Approximatifs (TEA) est un formalisme mathématique généralisant la théorie des ensembles classique Pawlak (2012). Cette théorie est fondée sur les notions d'indiscernabilité et d'approximation. Dans ce qui suit, nous présentons les notions essentielles de cette théorie pour la sélection des caractéristiques.

2.1 Système d'information

Le système d'information T permet de représenter les connaissances d'un domaine sous forme de $T = \{U, A\}$ où U correspond aux objets de l'univers et A les caractéristiques qui décrivent ces objets. A peut être divisée en deux sous-ensembles C et D appelés caractéristiques conditionnelles (toutes les caractéristiques) et la caractéristique de décision (la classe). Dans certains systèmes d'information, plusieurs caractéristiques de décision peuvent être présentées.

2.2 La relation d'indiscernabilité

Littéralement le mot indiscernable signifie que certaines choses de même nature peuvent être similaires sans être forcément identiques. D'après la théorie d'approximation, les objets indiscernables (ou similaires) sont caractérisés par les mêmes valeurs de certaines caractéristiques. Autrement dit, pour tout sous ensemble B de l'ensemble des caractéristiques de A, la relation $IND(B)$ est définie par : le couple $(x, y) \in IND(B)$, si et seulement si $a(x) = a(y)$ pour tout élément $a \in A$, avec $a(x)$ qui représente la valeur des caractéristiques a pour l'élément x. $IND(B)$ est une relation d'équivalence.

2.3 Les approximations de la théorie des ensemble approximatifs

Les approximations permettent d'associer à n'importe quel ensemble discernable B une paire d'ensembles - appelés approximation inférieure ($\underline{B}$) et approximation supérieure ($\overline{B}$). Dans l'approximation inférieure, ($\underline{B}$) est une description des objets qui appartiennent certainement à l'ensemble B. Elle est représentée par l'union de toutes les classes d'équivalence qui sont contenues dans l'ensemble cible tel que : $\underline{B} = \bigcup X \in U\{X_i \in U/[X_i]_B \subseteq B\}$. Tandis que l'approximation supérieure ($\overline{B}$) est une description des objets qui peuvent appartenir à l'ensemble B. Elle est représentée par l'ensemble de l'union de toutes les classes d'équivalence qui ont une intersection non vide avec l'ensemble cible tel que : $\overline{B} = \bigcup X \in U\{X_i \in U/[X_i]_B \cap B \neq \emptyset\}$.

2.4 Sélection des caractéristiques

Pour accomplir la tâche de sélection de caractéristiques certains concepts doivent être définis : La TEA définie la région positive $POS_C(D) = \bigcup \overline{C}(X)$ qui est l'ensemble des éléments

de U qui sont certainement classifiés aux partitions $U/IND(D)$ en se basant sur C. La dépendance des caractéristiques qui est défini par $k = \gamma(C, c_i) = \frac{|POS_C(c_i)|}{|U|}$ mesure le degré de dépendance k d'une caractéristique c_i par rapport à un ensemble des caractéristiques C. Sur ces bases, pour la sélection des caractéristiques, l'ensemble $R \subseteq C$ est dit un D-*reduct* de C si $\gamma(C, R) = \gamma(C)$ et il n'ya aucun $R' \subset R$ tel que $\gamma(C, R^{'}) = \gamma(C, R)$. En d'autres termes, le $Reduct$ est l'ensemble minimal de caractéristiques sélectionnées conservant le même degré de dépendance que l'ensemble des caractéristiques. La TEA peut générer un ensemble de reducts, $RED_D^F(C)$ et dans ce cas n'importe quel reduct de $RED_D^F(C)$ peut être choisi pour remplacer le système d'information initial.

3 Solution proposée

Dans cette section et dans le contexte des données massives, nous présentons notre nouvel algorithme distribué nommé "Sp-RST" qui est basé sur la TEA pour la sélection des caractéristiques. Notre algorithme est caractérisé par une implémentation distribuée basée sur l'écosystème Scala/Apache Spark Shanahan et Dai (2015). Le fonctionnement général du Sp-RST est présenté par la Figure 1.

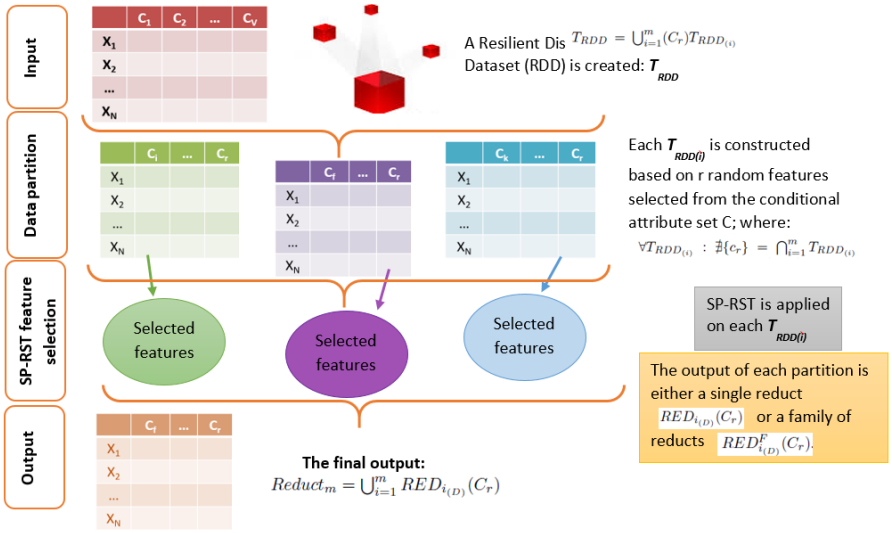

FIG. 1 – *Le fonctionnement général du SP RST.*

3.1 Formalisation générale

Sp-RST défini les données massives représentant le problème d'apprentissage comme un système d'information T_{RDD} où l'univers $U = \{x_1, \ldots, x_N\}$ est l'ensemble des objets, $C = \{c_1, \ldots, c_V\}$ est l'ensemble des caractéristiques conditionnelles parmi lesquelles Sp-RST sélectionne les caractéristiques les plus pertinentes et la caractéristique de décision $D =$

$\{d_1, \ldots, d_W\}$ correspond à la classe. Afin d'assurer la scalabilité de notre algorithme, Sp-RST partage la T_{RDD} en m blocs de données basés sur des partitions de C. Par conséquent, $T_{RDD} = \bigcup_{i=1}^{m}(C_r)T_{RDD_{(i)}}$; où $r \in \{1, \ldots, V\}$. Chaque $T_{RDD_{(i)}}$ est construit en fonction de r caractéristiques aléatoirement sélectionnées à partir de C ; où $\forall T_{RDD_{(i)}} : \nexists\{c_r\} = \bigcap_{i=1}^{m} T_{RDD_{(i)}}$. De ce fait, au lieu d'appliquer Sp-RST (voir Algorithme 1) sur la T_{RDD} comprenant l'ensemble des caractéristiques C, l'algorithme distribué sera appliqué sur chaque $T_{RDD_{(i)}}$. Par conséquent, nous pouvons palier aux limites de la TEA et garantir l'applicabilité de Sp-RST à un nombre de caractéristiques conséquent.

Algorithm 1 Sp-RST

Inputs : T_{RDD} Système d'information ; m nombre de partitions ; N nombre d'itérations
Output : $Reduct$

1: Calculer $IND(D)$
2: **for** each iteration $n \in [1, \ldots, N]$ **do**
3: Générer $T_{RDD_{(i)}}$ en se basant sur les m partitions
4: **for** each $T_{RDD_{(i)}}$ partition, $i \in [1, \ldots, m]$ **do**
5: Générer $AllComb_{(C_r)}$; Calculer $IND(AllComb_{(C_r)})$
6: Calculer $DEP(AllComb_{(C_r)})$; Séléctionner $DEP_{max}(AllComb_{(C_r)})$
7: Filtrer $DEP_{max}(AllComb_{(C_r)})$; Filtrer $NbF_{min}(DEP_{max}(AllComb_{(C_r)}))$
8: **end for**
9: **for** each $T_{RDD_{(i)}}$ output **do**
10: $Reduct_m = \bigcup_{i=1}^{m} RED_{i_{(D)}}(C_r)$
11: **end for**
12: **end for**
13: **return** ($Reduct = \bigcap_{n=1}^{N} Reduct_m$)

Afin de garantir d'avantage la performance de Sp-RST, nous appliquons l'algorithme N fois sur les m blocs de la T_{RDD}. Plus précisément, à travers toutes les itérations, l'algorithme générera d'abord les m $T_{RDD_{(i)}}$, ensuite pour chaque partition, les instructions distribuées de Sp-RST (Algorithme 1, lignes 5 à 7) seront exécutées à part la ligne 1 de l'algorithme. Cette tâche est indépendante des m partitions générées vu qu'elle calcule la relation d'indiscernabilité de la caractéristique de la décision $IND(D)$ et elle est non liée aux caractéristiques conditionnelles. En sortant de la boucle, ligne 9, le résultat de chaque partition est soit un seul reduct $RED_{i_{(D)}}(C_r)$ ou un ensemble de reducts $RED^{F}_{i_{(D)}}(C_r)$. En se basant sur les préliminaires de la TEA, tout reduct de $RED^{F}_{i_{(D)}}(C_r)$ peut être utilisé pour représenter la $T_{RDD_{(i)}}$. Par conséquent, si Sp-RST génère un seul reduct, pour une partition $T_{RDD_{(i)}}$ alors la sortie de cette phase de sélection de caractéristiques est l'ensemble des caractéristiques de $RED_{i_{(D)}}(C_r)$. Ces caractéristiques sont les plus pertinentes parmi l'ensemble C_r et résultant a un nouveau système d'information $T_{RDD_{(i)}}$, $T_{RDD_{(i)}}(RED)$, qui préserve quasiment la même qualité des données que $T_{RDD_{(i)}}(C_r)$ qui est basé sur tout l'ensemble des caractéristiques C_r. Si Sp-RST génère une famille de reducts alors l'algorithme choisit aléatoirement un reduct de $RED^{F}_{i_{(D)}}(C_r)$ pour représenter la $T_{RDD_{(i)}}$. À ce stade, à chaque bloc de données i correspond un ensemble de caractéristiques sélectionnées $RED_{i_{(D)}}(C_r)$. Cependant, puisque chaque $T_{RDD_{(i)}}$ est basé sur des caractéristiques distinctes, une union des caractéristiques sé-

lectionnées est nécessaire pour représenter la T_{RDD} initiale (Algorithme 1, lignes 9 à 11). L'algorithme est itéré N fois générant N $Reduct_m$. Ainsi, à la fin, une intersection de tous les $Reduct_m$ obtenus est nécessaire (Algorithme 1, ligne 13). Dans ce qui suit, nous allons élucider les 7 tâches distribuées de Sp-RST.

3.2 Détails algorithmiques

Tout d'abord, Sp-RST calcule la relation d'indiscernabilité pour la classe $D = \{d_1, \ldots, d_W\}$ (Algorithme 2) ; définie par $IND(D) : IND(d_i)$. Plus précisément, Sp-RST calcule la relation d'indiscernabilité pour chaque classe d_i en rassemblant les mêmes objets de T_{RDD} qui sont définis dans l'univers $U = \{x_1, \ldots, x_N\}$ ayant la même classe d_i. Pour ce faire, Sp-RST exécute l'opération $foldByKey$, où la classe d_i définit la clé et les identifiants des objets de T_{RDD}, id_i de x_i, définissent la valeur. L'ensemble des objets génèrés est retenu vu qu'il représente $IND(D) : IND(d_i)$.

Algorithm 2 Calculer $IND(D)$

Input : T_{RDD}
Output : $IND(D) : Array[Array[x_i]]$
1: $IND(d_i) = data.map\{case(id_i, vector, d_i) => (d_i, ArrayBuffer(id_i))\}$
$.foldByKey(ArrayBuffer.empty[Long])(\_ + + = \_)$
2: $IND(d_i).map\{case(d_i, x_i) => x_i\}.collect$

Puis, pour une partition spécifique, Sp-RST crée toutes les combinaisons possibles des caractéristiques C_r : $AllComb_{(C_r)} = C_r.flatMap(C_r.combinations)\ .drop(1)$ et calcule la relation d'indiscernabilité pour chaque combinaison générée (voir Algorithme 3). Sp-RST vise à regrouper tous les identifiants id_i des objets ayant la même combinaison des caractéristiques extraites $AllComb_{(C_r)}$. Pour ce faire, l'opération $foldByKey$ est utilisée où la combinaison des caractéristiques définie la clé et l'id_i comme valeur.

Algorithm 3 Calculer $IND(AllComb_{(C_r)})$

Inputs : T_{RDD_i}, $AllComb_{(C_r)}$
Output : $IND(AllComb_{(C_r)}) : Array[Array[id_i]]$
1: $IND(AllComb_{(C_r)})$ = $data.map\ \{case(id_i, vector, d_i) => ((AllComb_{(C_r)_i}, vector), ArrayBuffer(id_i))\}.foldByKey(ArrayBuffer.\ empty[Long])(\_ + + = \_)$
2: $IND(AllComb_{(C_r)}).map\{case(ListValues, id_i) => id_i\}.collect$

Dans Algorithme 4, les degrés de dépendances $\gamma(C_r, AllComb_{(C_r)})$ de chaque combinaison de caractéristiques sont calculés. Pour ce faire, les relations d'indiscernabilités calculées $IND(D)$ et $IND(AllComb_{(C_r)})$ ainsi que l'ensemble de toutes les combinaisons de caractéristiques $AllComb_{(C_r)}$ sont requis. La tâche consiste à tester, d'abord, si l'intersection de chaque $IND(d_i)$ avec chaque $IND(AllComb_{(C_r)})$ conserve toutes les caractéristiques de ce dernier (l'approximation inférieure). Si c'est le cas, un score qui est égal au nombre de caractéristiques dans $IND(AllComb_{(C_r)})$ est donné, zéro sinon. Étant donné que cette tâche est effectuée de façon distribuée où chaque machine traite certaines combinaisons

des caractéristiques, une première sommation des scores $IND(d_i)$ est opérée suivie d'une deuxième sommation pour avoir tous les scores $IND(D)$; référant au degré de dépendance $\gamma(C_r, AllComb_{(C_r)})$.

Algorithm 4 Générer $DEP(AllComb_{(C_r)})$

Inputs : $AllComb_{(C_r)}$, $IND(D)$, $IND(AllComb_{(C_r)})$
Outputs : $\gamma(C_r, AllComb_{(C_r)})$, $Size_{(AllComb_{(C_r)})}$

```
1: for i := AllComb_(C_r) do
2:     for j := IND(D) do
3:         for v := IND(AllComb_(C_r)) do
4:             if j.intersect(v).length == v.length then v.length
5:             else 0
6:         Reduce(_ + _)
7:     Reduce(_ + _)
```

Le résultat de cette étape est l'ensemble des degrés de dépendance $\gamma(C_r, AllComb_{(C_r)})$ des combinaisons des caractéristiques $AllComb_{(C_r)}$ et leurs tailles associées $Size_{(AllComb_{(C_r)})}$. À ce stade, Algorithme 5, Sp-RST recherche la valeur de dépendance maximale parmi tous les $\gamma(C_r, AllComb_{(C_r)})$.

Algorithm 5 Séléctionner $DEP_{max}(AllComb_{(C_r)})$

Input : RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(C_r, AllComb_{(C_r)})$]
Output : $MaxDependency$

```
1: RDD.max()(Ordering[Int].on(_._3))._3
```

La sortie $MaxDependency$ reflète d'une part le degré de dépendance de tout l'ensemble des caractéristiques (C_r) représentant T_{RDD_i} et d'autre part le degré de dépendance de toutes les combinaisons possibles des caractéristiques satisfaisant la contrainte $\gamma(C_r, AllComb_{(C_r)}) = \gamma(C_r)$. $MaxDependency$ est le seuil pour la sélection des caractéristiques.

Algorithm 6 Filtrer $DEP_{max}(AllComb_{(C_r)})$

Inputs : RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(C_r, AllComb_{(C_r)})$], $MaxDependency$
Output : Filtered-RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(C_r, AllComb_{(C_r)})$]

```
1: RDD.filter(_._3 == maxDependancy)
```

Ensuite, Sp-RST cherche l'ensemble de toutes les combinaisons ayant les mêmes degrés de dépendance que $MaxDependency$; $\gamma(C_r, AllComb_{(C_r)}) = MaxDependency$ en appliquant une fonction de filtrage ; Algorithme 6. A ce stade, Sp-RST supprime à chaque niveau de calcul les caractéristiques inutiles qui peuvent affecter négativement la performance de tout algorithme d'apprentissage.

Enfin, Algorithme 7, Sp-RST conserve l'ensemble des combinaisons ayant le nombre minimum des caractéristiques, $Size_{(AllComb_{(C_r)})}$, en appliquant une opération de filtrage et en satisfaisant les contraintes de reduct discutées précédemment ; $\gamma(C_r, AllComb_{(C_r)}) = \gamma(C_r)$ où

il n'ya aucun $AllComb^{'}_{(C_r)} \subset AllComb_{(C_r)}$ tel que $\gamma(C_r, AllComb^{'}_{(C_r)}) = \gamma(C_r, AllComb_{(C_r)})$.

Algorithm 7 Filtrer $NbF_{min}(DEP_{max}(AllComb_{(C_r)}))$

Input : RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(C_r, AllComb_{(C_r)})$]
Output : Filtered-RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(C_r, AllComb_{(C_r)})$]
1: $minNbF := RDD.min()(Ordering[Int].on(\_.\_2))._2$
2: $RDD.filter(\_.\_2 == minNbF)$

Chaque combinaison satisfaisant cette condition est considérée comme un ensemble minimal de caractéristiques les plus pertinentes décrivant l'ensemble des données de T_{RDD_i}.

3.3 Exemple de trace d'exécution

Dans cette section, nous allons présenter un exemple d'exécution de l'Algorithme 2 pour le calcul de la relation d'indiscernabilité de la classe. Supposant que nous avons le système d'information présenté par la Table 1 ayant comme classe $Flu = \{Yes, No\}$:

Patients	Headache	Muscle-pain	Temperature	Flu
o_1	Yes	Yes	very high	Yes
o_2	Yes	No	high	Yes
o_3	Yes	No	high	No
o_4	No	Yes	normal	No
o_5	No	Yes	high	Yes
o_6	No	Yes	very high	Yes

TAB. 1 – *Système d'information*

En se basant sur le partitionnement des données de MapReduce d'Apache Spark, les deux partitions sont les suivantes (Table 2 et Table 3) :

Patients	Headache	Muscle-pain	Temperature	Flu
o_1	Yes	Yes	very high	Yes
o_2	Yes	No	high	Yes
o_3	Yes	No	high	No

TAB. 2 – *Partition 1*

En appliquant la fonction Map de l'Algorithme 2 sur la première partition (Table 2), nous obtenons le résultat suivant :
— $<$"Yes", $O1>$
— $<$"Yes", $O2>$
— $<$"No", $O3>$
En faisant pareil pour la deuxième partition (Table 3), nous obtenons le résultat suivant :

Patients	Headache	Muscle-pain	Temperature	Flu
o_4	No	Yes	normal	No
o_5	No	Yes	high	Yes
o_6	No	Yes	very high	Yes

TAB. 3 – *Partition 2*

— $<$"No", $O4>$
— $<$"Yes", $O5>$
— $<$"Yes", $O6>$

Enfin, pour avoir la relation d'indiscernabilité de la classe $Flu = \{yes, no\}$ nous appliquons la fonction Reduce et plus précisément la fonction foldByKey pour obtenir le résultat suivant :

— $IND(Flu)$ = ["No", [{O3, O4}], ["Yes", [{O1, 02, O5, O6}]

Une fois l'indiscernabilité est calculée, SP-RST génère toutes les combinaisons possibles par partitions de caractéristiques qu'il a déjà créées. Par exemple :

— Partition X : {Headache}
— Output : {Headache}
— Partition Y : {Muscle-pain, Temperature}
— Output : {{Muscle-pain}, {Temperature}, {Muscle-pain, Temperature}}

Sur chaque élément des deux outputs des deux partitions X et Y, Sp-RST calclule les relations indiscernabilités (Algorithme 3). Pour ce faire, nous procédons comme présenté dans l'exemple de calcul de la relation indiscernabilité de la classe.

4 Expérimentations, Résultats et Analyse

4.1 Données et Implémentation

Pour démontrer la scalabilité de Sp-RST tout en palliant aux limites de la TEA et l'avantage de notre méthode de partitionnement en différents blocs, nous avons choisi le jeu de données Amazon Commerce Reviews Asuncion et Newman (2007). Cette base comprend 1 500 objets décrits à travers 10 000 caractéristiques et 50 classes distinctes. Les objets sont répartis d'une manière équitable sur les différentes classes, i. e., pour chaque classe, il y a 30 objets. Tous les tests ont été exécutés sur Grid5000 avec une configuration d'un processeur dual 8 core Intel Xeon E5-2630v3, 128 GB de mémoire. Nous étudions différents paramètres de Sp-RST qui est implémenté en Scala 2.11/Spark 2.1.1. Nous considérons d'abord les partitions 1000, 1200, 1250, 1330, 1500, 2000 et 2500, et nous les exécutons sur 1, 4, 8, 16 et 32 noeuds ; tous via 10 itérations. Nous avons utilisé Random Forest Prinzie et Van den Poel (2008) (org.apache.spark.mllib.tree.RandomForest) avec maxDepth = 6, numTrees = 300, featureSubjectStrategy = 'all 'et impurity =' gini', comme classifeur d'évaluation. Sur toutes les partitions générées, Random Forest est appliqué sur une taille de 70% comme ensemble d'apprentissage et un ensemble de test de 30%.

Notre objectif est de montrer que notre approche proposée Sp-RST est scalable et bien adaptée aux jeux de données ayant un grand nombre de caractéristiques. Ceci est obtenu sans introduire une perte d'informations significative.

4.2 Résultats et analyse

Pour monter l'efficacité de notre méthode et l'avantage de créer des blocs de caractéristiques, il faut d'abord monter sa scalabilité en terme de "Speed up" qui permet de mesurer le temps d'exécution en terme des noeuds utilisés, et en terme de temps d'exécution. Ce sont deux critères principaux pour l'évaluation de la scalabilité d'un algorithme massivement distribué. Une fois que nous avons montré la scalabilité de Sp-RST, il faut monter que l'algorithme n'engendre pas une perte d'information significative. Pour ce faire, il fallait analyser les résultats de classification de Random Forest sur la base Amazon composée de 10 0000 caractéristiques sans Sp-RST (partition1 : première colonne de Table 4) et avec Sp-RST sur les bases générées par toutes les partitions (le reste des colonnes de Table 4). Ceci est pour analyser également l'effet de cette partition sur les résultats de classification. Comme le Random Forest se base sur un processus aléatoire, il fallait passer par une étude statistique. Suite à 100 itérations, la moyenne, la médiane et l'écart type des résultats d'erreur de classification sont présentés dans la Table 4. En outre, pour étudier plus les résultats de classification entre Random Forest sur l'ensemble de données original et les ensembles réduits produits par Sp-RST nous effectuons le test de Wilcoxon Woolson (2008).

En terme de Speed up, nous gardons la taille de l'ensemble de données constante (où la taille est mesurée par le nombre des caractéristiques dans chaque partition) et nous augmentons le nombre de noeuds. À partir de la figure 2, nous observons que notre méthode a un bon Speed up pour les petites partitions. Plus la taille des données, i. e., le nombre de caractéristiques par partition augmente, plus le Speed up devient plus linéaire. Cela s'explique par le fait que moins de partitions impliquent que chaque partition possède plus de caractéristiques.

Comme discuté précédemment, le temps d'exécution augmente de façon exponentielle en fonction du nombre de caractéristiques et, par conséquent, l'utilisation d'un nombre plus important de noeuds est plus avantageux dans ce cas-là. Nous obtenons un bon Speed up si le nombre de caractéristiques par partition est compris entre 7 et 10 (1000 à 1330 partitions), mais pour 2000 et 2500 partitions le Speed up stagne rapidement.

Cette conclusion est également observée pour les temps d'exécutions. À partir de la figure 3, nous observons que pour quelques partitions, le temps d'exécution diminue rapidement en augmentant le nombre des noeuds, tandis que pour de nombreuses partitions, nous n'observons pratiquement aucune amélioration.

De ce fait, nous observons qu'il existe un compromis entre le nombre de partitions et le nombre de noeuds utilisés. Si quelques noeuds sont disponibles, il est conseillé d'utiliser un plus grand nombre de partitions pour réduire le temps d'exécution alors que le nombre de partitions devient moins important si l'on peut accorder un degré élevé de parallélisassion.

En terme de nombre de caractéristiques sélectionnées (voir Table 4), nous remarquons que le nombre de caractéristiques sélectionnées varie considérablement selon le nombre des partitions utilisées. Il est important de clarifier qu'une comparaison avec la version classique de sélection des caractéristiques basée sur la TEA n'est pas possible vu le nombre exponentiel de combinaisons de caractéristiques généré par la méthode comme expliqué dans l'introduction.

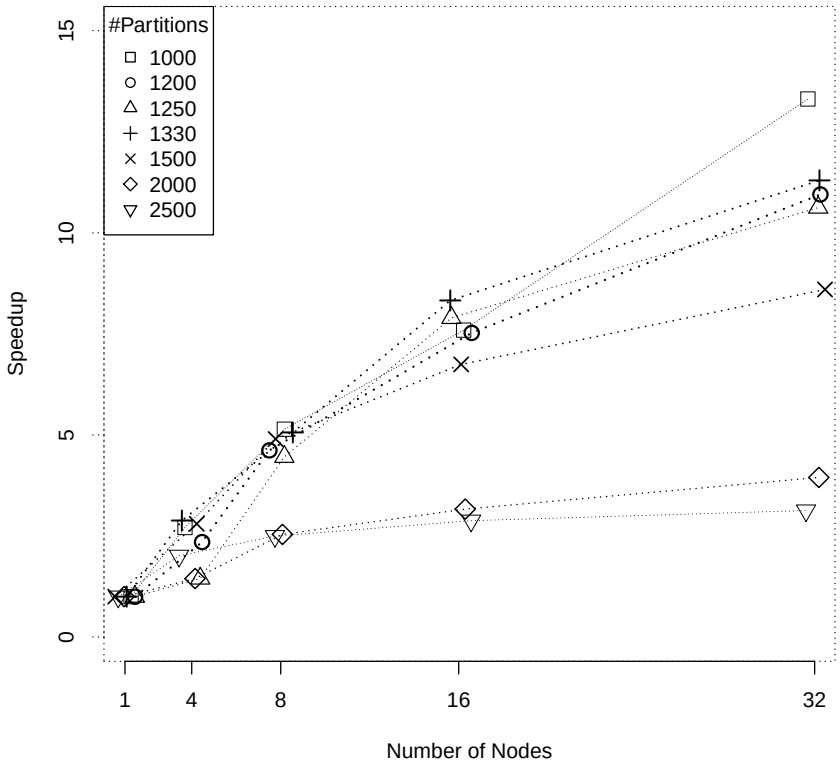

FIG. 2 – *Speedup pour 1 itération.*

Partitions	1	1000	1200	1330	2500
Moyenne	**50.50%**	51.00%	**49.69%**	**47.14%**	**45.16%**
Médiane	**52.53%**	**51.38%**	**50.77%**	**46.59%**	**45.40%**
Ecart Type	11.09%	11.13%	9.98%	11.77%	10.36%
p-valeur	-	0.4897	0.1852	0.009471	0.000485
Caractéristiques séléctionnées	-	6184	3665	2490	3209

TAB. 4 – *Les valeurs statistiques pour l'erreur de classification et le nombre de caractéristiques séléctionnées par partition*

Il est aussi important de clarifier qu'une comparaison avec d'autres méthodes de sélection de caractéristiques fera l'objet d'un travail futur.

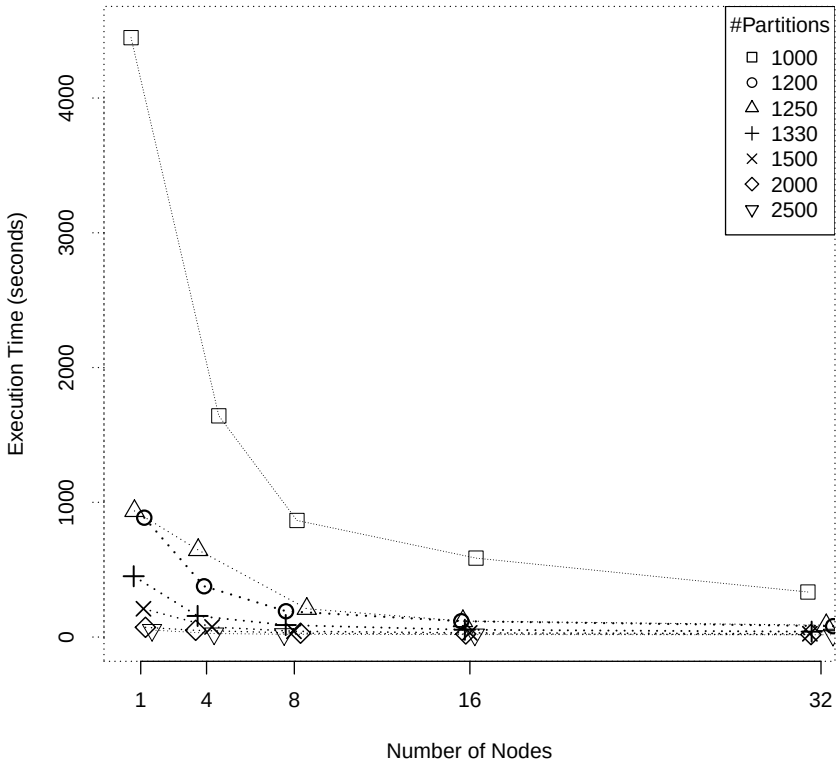

FIG. 3 – *Temps d'exécution pour 1 itération.*

En fonction de ces caractéristiques sélectionnées, nous avons analyser les résultats de classification de Random Forest. En terme d'erreur de classification et en utilisant Random Forest, nous présentons les résultats de 100 exécutions sur la base comportant les 10 000 caractéristiques et les jeux de données dérivés par Sp-RST avec quelques partitions (voir Table 4). Il est important de noter que malgré l'erreur de classification de Randon Forest semble élevée il convient de noter que Amazon contient 50 classes distinctes. Ainsi, une classification naïve bayésienne aurait une erreur de classification de 98%, ce qui est nettement plus élevé que notre approche. La médiane des erreurs de classification de Random Forest sur l'ensemble des données originales (sans Sp-RST) est supérieure à la valeur correspondante pour tous les paramètres de Sp-RST. Compte tenu des moyennes, pour les 1000 partitions il n'y a pas une grande différence par rapport au Random Forest sans Sp-RST. Ceci est aussi prouvé par la "p-valeur" avec un intervalle de confiance de 0.05. Suite à cette comparaison empirique, nous pouvons conclure que Sp-RST ne présente pas de perte d'information significative. Ce qui indique que notre approche est performante.

5 Conclusion

Nous avons présenté un nouvel algorithme distribué basé sur la théorie des ensembles approximatifs pour la sélection des caractéristiques pour les données massives. Nous avons montré que notre algorithme est scalable et capable de sélectionner les caractéristiques sans perte d'information significative. Une analyse de la paramètrisation de Sp-RST représente notre futur travail.

Acknowledgment

This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 702527.

Références

Asuncion, A. et D. Newman (2007). Uci machine learning repository.

Fan, W. et A. Bifet (2013). Mining big data : current status, and forecast to the future. *ACM sIGKDD Explorations Newsletter 14*(2), 1–5.

Labrinidis, A. et H. V. Jagadish (2012). Challenges and opportunities with big data. *Proceedings of the VLDB Endowment 5*(12), 2032–2033.

Pawlak, Z. (2012). *Rough sets : Theoretical aspects of reasoning about data*, Volume 9. Springer Science & Business Media.

Pawlak, Z. et A. Skowron (2007). Rudiments of rough sets. *Information sciences 177*(1), 3–27.

Prinzie, A. et D. Van den Poel (2008). Random forests for multiclass classification : Random multinomial logit. *Expert systems with Applications 34*(3), 1721–1732.

Shanahan, J. G. et L. Dai (2015). Large scale distributed data science using apache spark. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2323–2324. ACM.

Thangavel, K. et A. Pethalakshmi (2009). Dimensionality reduction based on rough set theory : A review. *Applied Soft Computing 9*(1), 1–12.

Woolson, R. (2008). Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*.

Summary

In this paper, we present a novel distributed rough set theory based algorithm for large-scale data pre-processing. Our experimental results show the scalability of our solution and its efficient applicability to Big Data without any significant information loss.